# Compte en banque : Exemple de conception et implantation par raffinement successif

11 octobre 2014

## 1 Énoncé

On veut écrire un programme qui fait la mise à jour de votre compte en banque. Le programme traite les transactions (dépôt (d), retrait (r) et terminer (t)) une à la fois, ajuste le solde et garde trace du nombre de transactions de chaque type. À la fin on imprime le solde de départ, le nouveau solde et le nombre de transactions de chaque type.

On doit refuser un retrait si le solde devient négatif.

On veut valider la lecture. Si après un certain nombre d'essais les données sont invalides, on arrête de traiter les transactions.

### 1.1 Analyse globale

**Entrée :**

1. (clavier) Solde de départ (réel)
2. Suite de transactions comprenant :
   (a) (clavier) code (caractères)
   (b) (clavier) montant (réels).

**Sortie :**

1. (écran) Solde de départ (réel)
2. (écran) Solde courant (réel)
3. (écran) Nombre de dépôts (entier)
4. (écran) Nombre de retraits (entier)

**Formules :**

1. Solde = Solde précédent + montant du dépôt
2. Solde = Solde précédent - montant du retrait

**Constantes :**

1. Dépôt = 'd'
2. Retrait = 'r'
3. Terminer = 't'

## 2 Conception globale : diagramme structurel

Notre diagramme structurel se divise en quatre niveaux. Les niveaux deux et trois comportent chacun trois modules.

Le premier niveau comprend seulement le module principal. Ce modules est ensuite divisé en trois modules dont le plus important est celuit qui traite toutes les transactions. Ce dernier module se décompose de nouveau en plusieurs modules : un pour la lecture, un pour traiter une seule transaction et un pour imprimer un résumé du traitement. Tous ces modules sont appelés jusqu'à ce que toutes les transactions soient traitées.

On applique la méthode de conception descendante par raffinement successif. Nous allons donc faire l'analyse, la conception et l'implantation du module du niveau 1. Une fois cela terminé, nous ferons l'Analyse, la conception et l'implantation des modules du niveau 2. Finalement, quand tous les modules du niveau deux serotn terminés, nous ferons l'analyse, la conception et l'implantation des modules du niveau 3.

Les modules de lecture retourne un code d'erreur si il y a eu un problème lors de la lecture. On traite les transactions seulement si la lecture initiale s'est faite correctement. Cela ne se reflète pas dans le diagramme.

Le diagramme structurel ressemble à ce qui suit :

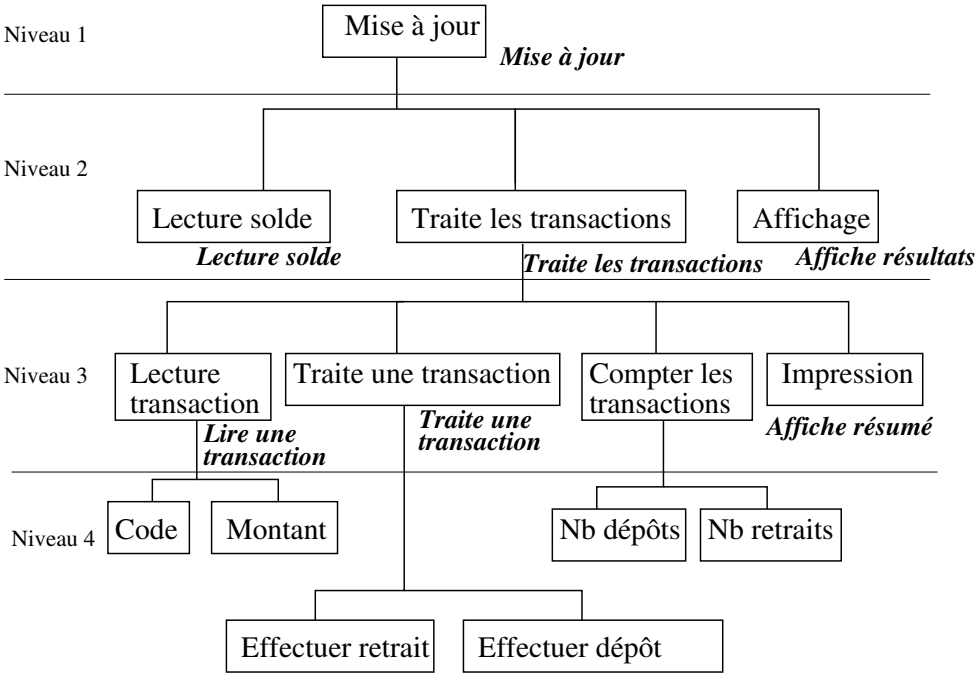


FIGURE 1 – Diagramme structurel

# 3 Module *Mise à jour* (principal)

## 3.1 Analyse

**Entrée :** rien
**Sortie :** rien

## 3.2 Conception (algorithme) :

1. Lecture du solde initial (module *Lecture solde*)
2. Selon le résultat de la lecture
   (a) Si lecture correcte
      i. Traite toutes les transactions (module *Traite transactions*)
      ii. Impression des résultats (module *Affiche résultats*)
   (b) Si lecture incorrecte → message d'erreur

## 3.3 Implantation

```
/*************************************************************/
/*  Compte.cc                                                */
/*                                                           */
/*  Ce programme fait la mise a jour mensuel d'un compte en */
/*  banque                                                   */
/*                                                           */
/*  Auteur : Gabriel Girard                                  */
/*                                                           */
/*  Entree :                                                 */
/*  Sortie :                                                 */
/*************************************************************/
#include <iostream>

using namespace std;

int main()
{
  // Fonctions utiles
  int lecture_solde(float &);
  int traite_transactions(float, float &, int &, int &);
  void affiche_resultat(float, float, int, int);

  // Definition des variables
  int nb_depot = 0, nb_retrait = 0;
  int code;                 // pour les codes d'erreurs

  float solde_depart, solde_courant;

  // on lit le solde de depart
  code = lecture_solde(solde_depart);

  // on traite toutes les transactions
```

```
  if (code == 0)
     {
       // on traite toutes les transactions
       code = traite_transactions(solde_depart, solde_courant,
                                  nb_depot, nb_retrait);

      // on affiche le resume des transactions et le solde
      affiche_resultat(solde_depart, solde_courant,
                       nb_depot, nb_retrait);
     }

  else cout << "Erreur lors de la lecture --- "
            << "traitement annule\n";
  return 0;
}
```

# 4 Module *Lecture solde*

## 4.1 Analyse

**Entrée :**

1. (clavier) solde initial (réel)

**Sortie :**

1. (paramètre) solde initial (réel)
2. (retour) code d'erreur ( 0 - correct, 1 - montant invalide)

## 4.2 Conception (algorithme)

On peut donner à l'utilisateur trois chances pour entrer des données valides. Après ces trois chances le module retourne un code d'erreur.

1. Pour chaque lecture (fin sur solde valide ou maximum 3)
   (a) Lecture du solde initial
   (b) Validation du solde
      i. si valide → ok (fin boucle)
      ii. si invalide → erreur

## 4.3 Implantation

```
/*********************************************************/
/*  Lecture_solde.cc                                     */
/*                                                       */
/*  Cette fonction lit le solde initial pour le compte   */
/*                                                       */
/*  Auteur : Gabriel Girard                              */
/*                                                       */
/*  Sortie : solde_depart (float)                        */
/*           code (int)                                  */
/*********************************************************/
```

```
#include <iostream>

int lecture_solde(float & solde_depart)
{
  // definition de la constante
  const int NB_ESSAI = 3;
  // Definition des variables
  int code=0, cpt=0;              // pour les codes d'erreurs
  bool valide = false;

  while (cpt < NB_ESSAI && !valide)
  {
    cout << "Entrer le solde de depart (il doit etre > 0) : ";
    cin  >> solde_depart;
    if (solde_depart > 0) valide = true;
    cpt++;
  }
  if (cpt >= NB_ESSAI) code = 1;

  return code;
}
```

# 5 Module *Traite les transactions*

## 5.1 Analyse

**Entrée ;**

1. (paramètre) solde initial (réel)

**Sortie :**

1. (paramètre) solde courant (réel)
2. (paramètre) nombre de dépôts (entier)
3. (paramètre) nombre de retraits (entier)

**Constantes :**

1. Dépôt = 'd'
2. Retrait = 'r'
3. Terminer = 't'

## 5.2 Conception (Algorithme)

1. Pour chacune des transactions
   (a) Lecture de la transaction (module *Lire une transaction*)
   (b) Selon le résultat de la lecture
      i. Si lecture correcte
         A. Traitement de la transaction (module *Traite une transaction*)
         B. On compte le nombre de transactions : dépôt et retrait
         C. Impression du nouveau solde (module *Affiche résumé*)
      ii. Si lecture incorrecte afficher un message d'erreur

## 5.3 Implantation

```
/*************************************************************/
/*  traite_les_transactions.cpp                              */
/*                                                           */
/*  Ce programme traite toutes les transactions sur un compte */
/*  en banque.  Il manipule par l'intermediaire de fonctions  */
/*  les donnees suivantes:                                   */
/*               solde (float)                               */
/*               solde_cour (float)                          */
/*               nb_dep, nb_ret (int)                        */
/*   Entree :    solde (float)                               */
/*   E/S         solde_cour (float)                          */
/*               nb_dep, nb_ret (int)                        */
/*                                                           */
/*  Auteur : Gabriel Girard                                  */
/*************************************************************/
#include <iostream>

traite_les_transactions(float solde, float& solde_cour,
                    int& nb_dep, int& nb_ret)
{
  // Fonctions utiles
  int lire_trans(char &, float &);
  int traite_une_trans(float, char, float, float &);
  void affiche_resume(float, char, float, float);

  // Definition des variables
  char code_trans;

  int code;                  // pour les codes d'erreurs
  float montant;

  solde_cour = solde;

  // on lit la transaction
  code = lire_trans(code_trans, montant);

  while ((code == 0) && (code_trans != 't'))
  {
     // on traite une transaction
     traite_une_trans(solde, code_trans,
                      montant, solde_cour);

     // mise a jour du nombre de depots ou retraits
     if (code_trans == 'r' && solde != solde_cour) nb_ret++;
```

```
     else nb_dep++;

     // on affiche le resume
     affiche_resume(solde, code_trans,
                    montant, solde_cour);

     // mise a jour du solde
     solde = solde_cour;

     // on lit la prochaine transaction
     code = lire_trans(code_trans, montant);

  }

  if (code != 0)
           cout << "\n\t *** Mauvaise transaction --- "
                << "traitement interrompu ***\n";
  else     cout << "\nFin du traitement des transactions\n"
                << "---------------------------------" << endl;

  return code;
}
```

# 6 Module *Affiche résultats*

Ce module est simple... On saute exceptionnellement l'analyse/conception pour passer directement à l'implantation.

```
/*********************************************************/
/*  Cette fonction imprime le resume des transactions      */
/*                                                         */
/*  Entree et sortie : solde_depart (float)                */
/*                     solde_courant (char)                */
/*                     nb_depots (int)                     */
/*                     nb_retraits (int)                   */
/*                                                         */
/*********************************************************/
#include <iostream>

void affiche_resultat(float solde1, float solde2,
                      int   nb_dep, int   nb_ret)
{
   cout << "Resume des transactions du mois. " << endl;
   cout << "Solde initial      ==> " << solde1 << endl;
   cout << "Solde final        ==> " << solde2 << endl;
   cout << "Nombre de depots   ==> " << nb_dep << endl;
   cout << "Nombre de retraits ==> " << nb_ret << endl;
}
```

# 7 Module *Lire une transaction*

Le premier module du niveau 2 à analyser est celui de lecture.

## 7.1 Analyse

**Entrée :**

1. (clavier) code de transaction (caractère)
2. (clavier) montant (réel)

## 7.2 Conception (algorithme)

1. Pour chaque lecture (fin sur code valide ou un nombre d'essais maximal)
   (a) lire le code
   (b) Selon la validité du code
      i. Si code valide → lire montant
      ii. si code invalide → erreur

## 7.3 Implantation

```
/*************************************************************/
/*  Lire_une_transaction.cc                                  */
/*                                                           */
/*  Cette fonction lit la prochaine transaction              */
/*                                                           */
/*  Sortie : code_trans (char)                               */
/*           montant (float)                                 */
/*                                                           */
/*************************************************************/
#include <iostream>

int lire_trans(char& code_trans, float& montant)
{
  // definition de la constante
  const int NB_ESSAI = 3;

  // Definition des variables

  int code=0, cpt=0;             // pour les codes d'erreurs
  bool valide = false;

  while (cpt < NB_ESSAI && !valide)
  {
    cout << "Entrer le code de la transaction\n";
    cout << "         r -- retrait\n";
    cout << "         d -- depot\n";
    cout << "         t -- pour terminer\n\n";
    cin  >> code_trans;
    if (code_trans == 'd' || code_trans == 'r'
                          || code_trans == 't')
```

```
        valide = true;
    else cout << "Code de transaction invalide; "
              <<  "Recommencer.\n";
    cpt++;
  }
  if (cpt < NB_ESSAI && code_trans != 't')
  {   cout << "Entrer le montant de la transaction : ";
      cin >> montant;
  }
  else if (cpt >= NB_ESSAI) code = 1;

  return code;
}
```

# 8 Module *Traite une transaction*

## 8.1 Analyse

**Entrée :**

1. (paramètre) code (caractère)
2. (paramètre) montant (réel)
3. (paramètre) solde de départ (réel)
4. (paramètre) solde (réel)

**Sortie :**

1. (paramètre) solde (réel)

**formule :**

1. Solde = Solde précédent + montant du dépôt
2. Solde = Solde précédent - montant du retrait

## 8.2 Conception (algorithme)

1. Selon le code
   (a) Si retrait effectuer retrait
   (b) Si dépôt effectuer dépôt

## 8.3 Implantation

```
/*************************************************************/
/*  traite_une_trans.cpp                                     */
/*                                                           */
/*  Cette fonction traite une transaction sur le compte      */
/*                                                           */
/*  Entree : solde_depart (float)                            */
/*           code_trans (char)                               */
/*           montant (float)                                 */
/*  Sortie : solde_courant (float)                           */
```

```
/*                                                            */
/************************************************************/
#include <iostream>

void traite_une_trans(float solde, char code_trans,
                      float montant, float& solde_courant)
{
   float temp;

   switch (code_trans) {

   case 'd' : solde_courant = solde + montant;
              break;

   case 'r' : temp = solde - montant;
              if (temp < 0)
                  cout << " ATTENTION -- solde negatif "
                       << "------ transaction refusee \n";
              else solde_courant = temp;
   }
}
```

# 9 Module *Affiche résumé*

```
/*********************************************************/
/*  Cette fonction imprime le resume de la transaction    */
/*                                                        */
/*  Entree : solde_depart (float)                         */
/*           code_trans (char)                            */
/*           montant (float)                              */
/*           solde_courant (float)                        */
/*********************************************************/
void affiche_resume(float solde1, char code_t,
                    float montant, float solde2)
{
  switch (code_t)
  {
    case 'r' : cout << "Retrait de $" << montant << endl;
               cout << "solde initial ==> " << solde1 << endl;
               cout << "solde final   ==> " << solde2 << endl;
               break;

    case 'd' : cout << "depot de $" << montant << endl;
               cout << "solde initial ==> " << solde1 << endl;
               cout << "solde final   ==> " << solde2 << endl;

   }
}
```